Qbox (qb@ll branch)

Summary Version

1.2

Purpose of Benchmark

Qbox is a first-principles molecular dynamics code used to compute the properties of materials directly from the underlying physics equations. Density Functional Theory is used to provide a quantum-mechanical description of chemically-active electrons and nonlocal norm-conserving pseudopotentials are used to represent ions and core electrons. The method has $O(N^3)$ computational complexity, where N is the total number of valence electrons in the system.

Characteristics of Benchmark

The computational workload of a Qbox run is split between parallel dense linear algebra, carried out by the ScaLAPACK library, and a custom 3D Fast Fourier Transform. The primary data structure, the electronic wavefunction, is distributed across a 2D MPI process grid. The shape of the process grid is defined by the nrowmax variable set in the Qbox input file, which sets the number of process rows. Parallel performance can vary significantly with process grid shape, although past experience has shown that highly rectangular process grids (e.g. 2048 x 64) give the best performance.

The majority of the run time is spent in matrix multiplication, Gram-Schmidt orthogonalization and parallel FFTs. Several ScaLAPACK routines do not scale well beyond 100k+ MPI tasks, so threading can be used to run on more cores. The parallel FFT and a few other loops are threaded in Qbox with OpenMP, but most of the threading speedup comes from the use of threaded linear algebra kernels (blas, lapack, ESSL).

Qbox requires a minimum of 1GB of main memory per MPI task.

Mechanics of Building Benchmark

To compile Qbox:

1. Go to `trunk/src`

2. Create an architecture include file, e.g. `bgq_ctf_essl.mk`. This file should define the make variables for compiling and linking (`CXX, CXXFLAGS, LDFLAGS`

etc.) and should contain the locations of all libraries and include files.

3. Qbox requires ScaLAPACK and dependent libraries (blas and lapack or vendor-equivalents, e.g. ESSL), FFTW 2.x or equivalent, and (optionally) Xerces XML.

4. Build the code using the `ARCH` variable to point to this include file, e.g.

```
make ARCH=bgq_ctf_essl
```

If the `ARCH` variable is not specified, a default value will be guessed from the `hostname` or `uname` commands listed in `Makefile.arch`.

5. Object files are stored in `objs-$(ARCH)` subdirectories, so one can compile multiple versions of the code simultaneously in the same directory.

Mechanics of Running Benchmark

To run Qbox, one needs an input file (.i), a coordinate file (.sys) and pseudopotential file(s) (.xml). To provide an adjustable set of inputs for performance testing, a flexible input generation script is provided in `examples/mgo_benchmark`. To use it, simply input the number of atoms and the target number of MPI tasks, e.g.

```
qbox_mgo_makeinputs.pl 8000 65536
```

This will generate .i and .sys files for the closest number of atoms that form a uniform rectangular crystal, which should be copied along with the provided `Mg.xml` and `O.xml` pseudopotential files to the run directory. Note that Qbox scales as $O(N^3)$, so larger systems will take significantly longer to run on the same number of tasks. On current machines, 4000-8000 atoms has been found to be tractable on 8k-128k tasks.

The input file can be specified either as an argument or as stdin to Qbox, as in the following examples (which use SLURM launch commands, where –n specifies the number of tasks):

```
srun -n 16384 ./qb-bgq_ctf_essl mgo.N1024.i > mgo.N1024.out
```

```
srun -n 64 ./qb-linux < sih4.i > sih4.out
```

The input generation script will try to choose a reasonable value for the process grid shape, but other values should be tested by changing the value of the `nrowmax` variable in the input (.i) file. ScaLAPACK performance can be further improved by manually tweaking the local data size used for the block-cyclic distribution. To do so, add one of the following lines just below the `nrowmax` line:

```
set matrix_loc 512 32
set pblock 8 4
```

The `matrix_loc` variable explicitly sets the local data size. The `pblock` variable chooses the local size which corresponds to the target number of cycles (e.g. for m = 10,485,760, nrowmax=1024, pblock=8, the local data size would be 1280). The default data layout is often the most efficient.

Sample command line parameters or inputs for:

1. Small problem: examples/sih4/sih4.i
2. Medium problem: mgo.N1024.i, generated with qbox_mgo_benchmark.pl

3. Large CORAL reference (FOM baseline) problem: mgo.N7936.i, generated with qbox_mgo_benchmark.pl

4. CORAL class problem: to get 4x more work, we need $4^{1/3}$ = 1.587 more atoms, mgo.N12544.i is the closest size.

Verification of Results

Timings are printed at the end of the run. The total scfloop time defines the overall performance:

```
<timing where="run"      name=" scfloop"    min="1372.748 " max="1373.275 "/>
```

The FOM for the Qbox benchmark is the number of scf iterations per second, multiplied by the total number of atoms cubed. The mgo input files use 5 scf iterations, so the time per scf iteration can be computed by taking the max scfloop time and dividing by 5, i.e. 1373.275/5 = 274.655 sec. This corresponds to 1/274.655 = 3.641E-03 iterations per second. For a 512 atom system, the FOM would thus be 3.621E-03*512^3 = 4.887E+05.

Check the output file for <WARNING> or <ERROR> tags, which may indicate a correctness problem. Confirm that reported energy values are finite (i.e. not nan) and that the reported total_electronic_charge is an integer value.